โครงการสุดท้าย (Final Project)

วิชา 204466 การเรียนรู้เชิงลึก (Deep Learning)

จัดทำโดย

อภิภู ชูเจริญประกิจ เลขประจำตัว 6510503891

อิทธิเดช นามเหลา เลขประจำตัว 6510503905

1. ชื่อหัวข้อ

(ภาษาไทย) แบบจำลอง GRU สำหรับคาดการณ์ราคา Bitcoin โดยใช้ข้อมูลมหภาคหลายสินทรัพย์และ Regime ของความผันผวน

(ภาษาอังกฤษ) GRU-based Model for Bitcoin Price Forecasting Using Multi-Asset Macroeconomic Data and Volatility Regimes

2. หัวข้อนี้น่าสนใจอย่างไร ทำไมถึงเลือกหัวข้อนี้มาทำเป็น final project

ผมเลือกหัวข้อนี้เพราะมันตรงกับโจทย์จริงของคนเล่นตลาด คืออยากรู้ว่าระยะสั้นต่อจากนี้ราคาจะเอียงไปทางไหน ทั้งที่ข้อมูลการเงินมันแกว่งแรงและเปลี่ยนโหมดบ่อย งานนี้เลยเหมาะจะลอง Deep Learning ที่อ่านลำดับเวลาได้ดี ผมดึงข้อมูลหลายสินทรัพย์จาก yfinance ทั้ง Bitcoin ดัชนีหุ้นสหรัฐ ทองคำ และ NASDAQ มาใช้ร่วมกัน เพื่อให้โมเดลมองบริบทข้ามตลาด ไม่ได้จ้องเฉพาะราคาเหรียญอย่างเดียว

จุดที่ผมชอบคือการทำให้โมเดล regime-aware ด้วยการคำนวณความผันผวน 30 วันของ BTC แล้วแบ่งเป็น low mid high ใส่เข้าไปเป็นฟีเจอร์เพิ่ม นี่ช่วยให้โมเดลรู้บรรยากาศตลาดก่อนตัดสินใจ จากนั้นผมสร้างฟีเจอร์เทคนิคที่คุ้นเคยอย่าง log return, volatility, RSI, moving average, MACD รวมถึงความสัมพันธ์ข้ามสินทรัพย์แบบ rolling correlation เพื่อให้ข้อมูลที่ป้อนเข้าโมเดลมีทั้งโมเมนตัม เสียงรบกวน และสัญญาณเชิงโครงสร้าง

ฝั่งโมเดล ผมใช้ GRU ล้วนๆ เพราะตรงกับลำดับเวลาและกินสเปคน้อยกว่า LSTM/Transformer ในโจทย์นี้ โค้ดวางเป็น sequence length 128 วัน ป้อนฟีเจอร์หลายมิติ เข้า GRU 3 ชั้น มี dropout กัน overfit แล้วต่อ Linear ออก logit เดียวสำหรับทิศทางล่วงหน้า H วัน ปรับด้วย AdamW และ CosineAnnealingLR มี early stopping กันหลุด นอกจากนี้ผมประเมินด้วย AUC, F1, Accuracy แบบ time-split และลองปรับ decision threshold บน validation เพื่อให้ผลใช้งานจริงบาลานซ์ขึ้น ไม่ได้ยึด 0.5 ตายตัว

ที่เลือกทำเป็นไฟนอลเพราะมันครบทั้ง data pipeline และ modeling workflow ตั้งแต่โหลดหลายแหล่ง ทำฟีเจอร์ข้ามตลาด ใส่ regime ให้โมเดลเข้าใจ state ของตลาด ไปจนถึงเทรน ประเมิน และจูน threshold ในแบบที่เอาไปใช้จริงได้ งานนี้เลยไม่ใช่แค่คะแนน แต่คือการประกอบร่างระบบพยากรณ์ทิศทางที่

อธิบายได้พอสมควร วัดผลชัดเจน และขยายต่อได้ เช่น เพิ่ม backtest หรือเพิ่ม uncertainty ภายหลังถ้าต้องการ ผมว่ามันตอบโจทย์ทั้งเชิงวิชาการและเชิงปฏิบัติของคอร์สนี้พอดีครับ

3. ทำไมหัวข้อนี้จึงต้องใช้ deep learning ในการแก้ปัญหา เปรียบเทียบกับการแก้ปัญหานี้ด้วยวิธีอื่นๆ วิธี deep learning มีข้อเด่น ข้อด้อยอย่างไร

ธรรมชาติของราคาตลาดเป็นลำดับเวลา มี non-linearity สูง และเปลี่ยนโหมดตลอด การใช้ Deep Learning แบบ GRU จึงเหมาะ เพราะมันอ่าน pattern ตามเวลาได้เอง ทั้งสัญญาณสั้นและแนวโน้มยาว โดยไม่ต้องกำหนดกฎตายตัว นอกจากนี้ผมป้อนฟีเจอร์หลายสินทรัพย์เข้าพร้อมกัน เช่น log return, volatility, RSI, MACD, สหสัมพันธ์ระหว่าง BTC กับ S&P500 ทอง และ NASDAQ รวมถึงตัวบ่งชี้ regime จาก realized vol 30 วันแบบ one-hot เพื่อให้โมเดลรับรู้บรรยากาศตลาดก่อนค่อยตัดสินใจ

เทียบกับวิธีอื่น เช่น rule-based technical indicator strategy เข้าใจง่ายแต่แข็งตัว ปรับตามตลาดยาก โมเดลสถิติอย่าง ARIMA หรือเส้นตรง โปร่งใสแต่สมมติฐาน stationarity/linear มักไม่ตรงของจริง ส่วน tree/XGBoost จับ non-linear ได้ดีบนตารางคงที่ แต่ไม่ได้อ่านลำดับเวลาโดยธรรมชาติ ต้องทำฟีเจอร์วิศวกรรมหนัก ฝั่ง Deep Learning จุดเด่นคือ เรียนรู้ไดนามิกของลำดับและความสัมพันธ์ข้ามสินทรัพย์แบบ end-to-end และรองรับฟีเจอร์ regime ได้ลื่นกว่า

ข้อด้อยของ Deep Learning คือ ต้องการข้อมูลและการควบคุม overfit มากกว่า ใช้ทรัพยากรคำนวณเยอะ และตีความยากกว่าวิธีคลาสสิก ผมเลยใช้ standardization, early stopping และประเมินผลด้วย AUC/F1/ACC รวมถึงวิเคราะห์ threshold ให้เหมาะกับวัตถุประสงค์ แม้ตอนนี้ยังไม่ได้ใส่ Attention หรือ MC Dropout ในโค้ด แต่สถาปัตยกรรม GRU + ฟีเจอร์หลายสินทรัพย์ + regime ก็เพียงพอให้เห็นประโยชน์ของ Deep Learning

สรุปคือ งานนี้เลือก Deep Learning เพราะโจทย์เป็นลำดับเวลาหลายสินทรัพย์ที่ความสัมพันธ์ไม่เป็นเส้นตรงและแปรผันตามภาวะตลาด GRU ช่วยเรียนรู้ temporal pattern ได้ตรงจริต ขณะที่ฟีเจอร์ cross-asset และ regime ทำให้โมเดลรู้บริบทตลาดมากขึ้น พูดง่ายๆ คือจากโค้ดชุดนี้ จุดขายอยู่ที่ GRU สำหรับ sequence จริงๆ บวกฟีเจอร์ macro/relational และ regime-aware ที่ประกอบกันแล้วให้สัญญาณที่ practical กว่าวิธีพื้นฐาน

4. อธิบายสถาปัตยกรรม deep learning ที่ใช้ (feedforward NN CNN RNN GAN หรือ VAE) วาดรูปแสดงจำนวนโหนด weight bias รวมถึงการเชื่อมต่อ และ activation function ต่างๆให้ชัดเจน

โมเดลที่ใช้คือ Recurrent Neural Network (RNN) แบบพิเศษที่เรียกว่า Gated Recurrent Unit (GRU) เพราะต้องการให้โมเดลจำ pattern ของราคาย้อนหลังหลายวันเพื่อที่จะดูแนวโน้มโดยรับอินพุตเป็น sequence length = 128 วัน และเพราะว่า GRU นั้นมีประสิทธิภาพใกล้เคียงกับ LSTM แต่ train เร็วกว่าใช้หน่วยความจำน้อยกว่า แต่ละวันเป็นเวกเตอร์ฟีเจอร์หลายสินทรัพย์ เช่น BTC S&P500 Gold NASDAQ

และฟีเจอร์เทคนิคอย่าง log-return volatility RSI moving average momentum MACD ความสัมพันธ์ข้ามสินทรัพย์ และตัวชี้วัด regime ผมสเกลฟีเจอร์ด้วย StandardScaler แล้วป้อนเป็นเทนเซอร์ขนาด batch จำนวนวันในลำดับ จำนวนฟีเจอร์ จากนั้นผ่าน GRU หลายเลเยอร์ hidden size 256 และ dropout จากนั้นดึง hidden ของเวลาแท่งสุดท้ายเข้าชั้น Linear หนึ่งโหนดเพื่อให้ได้ logit แล้วค่อยแปลงเป็นความน่าจะเป็นด้วย sigmoid ตอนประเมิน โมเดลเทรนด้วย BCEWithLogitsLoss และ AdamW

ชั้น GRU และพารามิเตอร์ภายใน

ในหนึ่งเลเยอร์ของ GRU แต่ละเวลา มีโหนดสถานะแอบแฝงขนาด 256 หน่วย และมีเกตสามชุดที่เรียนรู้พารามิเตอร์เองทั้งหมด

- **update gate z** ใช้ตัดสินว่าจะเก็บข้อมูลใหม่มากแค่ไหนจากอินพุตวันนี้เทียบกับความจำเดิมของเมื่อวาน
- **reset gate r** ใช้ควบคุมว่าควรลืมส่วนไหนของความจำเดิมก่อนคำนวณข้อมูลใหม่
- **candidate hidden** ใช้สร้างข้อเสนอค่าความจำตัวใหม่จากอินพุตวันนี้ร่วมกับความจำเดิมที่ถูกรีเซ็ตแล้ว

ทุกเกตมีน้ำหนักสองก้อน คือ น้ำหนักอินพุตไปซ่อน และน้ำหนักซ่อนกลับมาซ่อน พร้อมเวกเตอร์ไบแอสหนึ่งชุดต่อเกต รวมแล้วต่อเลเยอร์จะมีพารามิเตอร์สามชุดซ้ำแพทเทิร์นเดียวกัน กำหนดให้เรียนรู้จากข้อมูลตามลำดับเวลา ฟังก์ชันกระตุ้นที่ใช้ภายใน GRU คือ sigmoid สำหรับเกต และ tanh สำหรับค่าผู้สมัครของสถานะแอบแฝง

การเชื่อมต่อและมิติของข้อมูล

อินพุตหนึ่งวันคือเวกเตอร์ฟีเจอร์ขนาด F เท่ากับจำนวนฟีเจอร์ที่เลือกจริงจากตาราง เมื่ออ่านครบ 128 วันแล้วจะส่งเข้า GRU เลเยอร์แรกที่รับอินพุตมิติ F และปล่อยเอาต์พุตมิติ 256 สำหรับทุกเวลา เลเยอร์ถัดไปของ GRU รับมิติ 256 และปล่อย 256 ต่อเช่นกัน ตลอดความยาวซีเควนซ์ โมเดลจะส่งต่อ hidden state จากวัน t ไปยังวัน t+1 ด้วยน้ำหนักชุดเดียวกันในแต่ละเลเยอร์ เมื่อถึงวันสุดท้าย ผมดึงเวกเตอร์ hidden ของวันสุดท้ายขนาด 256 ไปเข้าชั้นเอาต์พุต

ชั้นเอาต์พุตและฟังก์ชันกระตุ้น

เอาต์พุตเป็นชั้น Linear หนึ่งโหนด มีน้ำหนักขนาด 256 คูณ 1 และไบแอสหนึ่งตัว ได้ค่า logit สำหรับการจำแนกไบนารี จากนั้นตอนวัดผลผมใช้ sigmoid แปลง logit เป็นค่าความน่าจะเป็นของทิศทางขาขึ้น แล้วค่อยตัดสินใจด้วย threshold ที่จูนบนชุด validation ระหว่างการรายงาน ผมแสดงทั้ง F1 AUC และ Accuracy เพื่อให้เห็นผลหลายมุม

การนับจำนวนพารามิเตอร์ให้ครบถ้วน

ถ้าอยากนับพารามิเตอร์ทั้งหมดอย่างเป็นระบบ ให้ตั้ง F เท่ากับจำนวนฟีเจอร์จริง และ H เท่ากับ 256

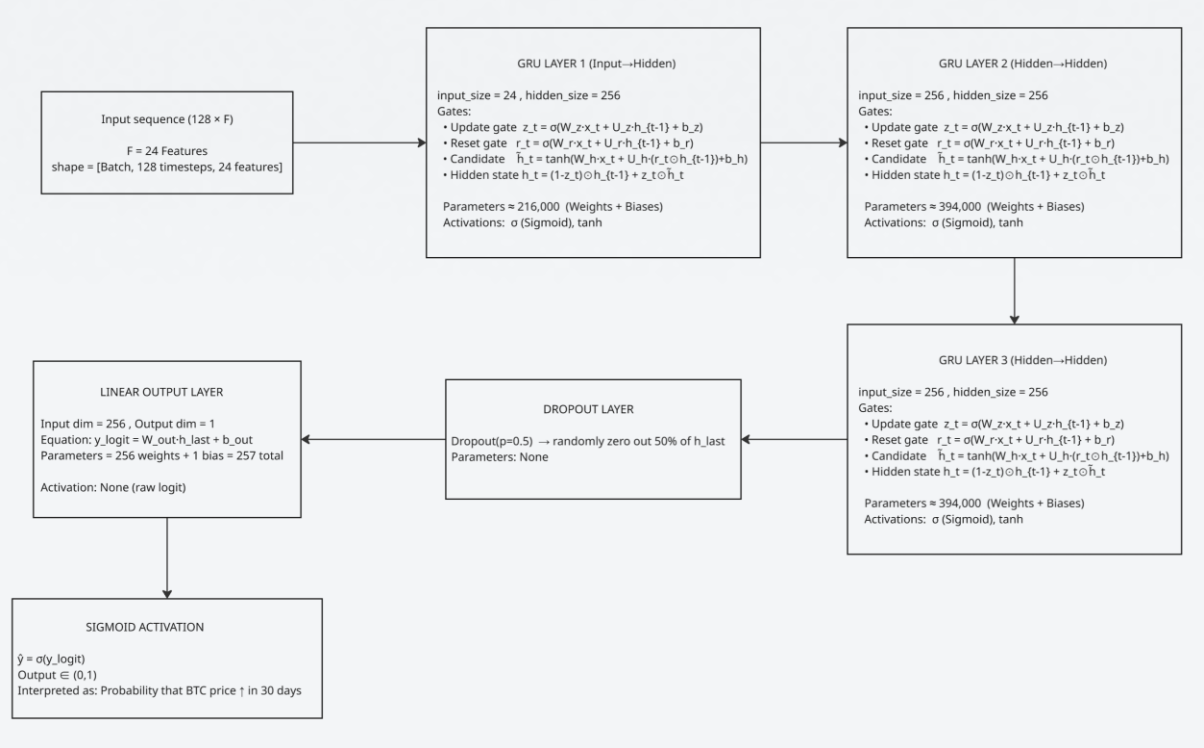
- GRU เลเยอร์แรก มีสามเกต แต่ละเกตมีน้ำหนักอินพุตขนาด F คูณ H น้ำหนักรีเคอร์เรนซ์ขนาด H คูณ H และไบแอส H ดังนั้นรวมเลเยอร์แรกเท่ากับสามคูณ วงเล็บ F คูณ H บวก H คูณ H บวก H
- GRU เลเยอร์ที่สองและสาม รับอินพุต H เช่นกัน จึงแต่ละเลเยอร์เท่ากับสามคูณ วงเล็บ H คูณ H บวก H

คูณ H บวก H

• ชั้น Linear สุดท้ายมีพารามิเตอร์ H บวก 1

เมื่อแทน H เท่ากับ 256 และแทน F ด้วยค่าจริงจาก len(features) ก็จะได้จำนวนพารามิเตอร์รวมของทั้งโมเดลอย่างชัดเจน

สรุปคือ โมเดลนี้ของผมคือ GRU แบบสามเลเยอร์ที่จดจำบริบทตามลำดับ 128 วัน ใช้ sigmoid และ tanh ภายในเกต เรียนรู้น้ำหนักและไบแอสครบทุกเกต เพื่อนำ hidden สุดท้ายไปตัดสินใจผ่าน Linear หนึ่งโหนด แล้วแปลงเป็นความน่าจะเป็นด้วย sigmoid ตอนประเมิน ตรงตามโค้ดที่กำหนดและสื่อโครงสร้างโหนด การเชื่อมต่อ น้ำหนัก ไบแอส และ activation function ครบถ้วนครับ



รูปภาพ แสดงการอธิบายสถาปัตยกรรม

5. อธิบายโค้ด PyTorch หรือ TensorFlow รวมไปถึงโค้ดส่วนอื่นๆที่ใช้ในการเชื่อมต่อกับโมเดลอื่นๆ อย่างชัดเจน ส่วนไหนจัดการกับข้อมูล ส่วนไหนสร้างโมเดล ส่วนไหน train ฯลฯ

Load Multi-Asset / Macro Data

```
def safe_load(symbol, start="2018-01-01"):
    data = yf.download(symbol, start=start)
    if isinstance(data, pd.DataFrame):
        close_col = data.columns[-1]  # ใช้คอลัมน์สุดท้าย (yfinance auto-adjust)
        return data[close_col].rename(symbol)
    return None
```

safe_load(symbol, start)

ดาวน์โหลดราคาย้อนหลังของสัญลักษณ์ที่กำหนดตั้งแต่วันที่ start หากได้ DataFrame จะเลือกคอลัมน์สุดท้าย (โดยปกติมักเป็นราคาปรับแล้ว) แล้วคืนเป็น Series ที่ตั้งชื่อคอลัมน์เท่ากับชื่อสัญลักษณ์ เพื่อเตรียมเอาไปต่อรวมกับตัวอื่นได้สะดวก

Feature engineering (Multi-asset + Regime)

```
def rsi_14(x: pd.Series):
    up = x.diff().clip(lower=0).rolling(14, min_periods=14).mean()
    dn = x.diff().clip(upper=0).abs().rolling(14, min_periods=14).mean()
    return 100 - (100/(1 + (up/dn)))
```

rsi_14(x: pd.Series)

คำนวณผลต่างรายวัน x.diff() แยกขาขึ้น up และขาลง dn

หา average gain/ loss แบบ rolling 14 วัน แล้วนำไปเข้าสูตร RSI (0–100)

```
new_features = {}
for c in base.columns:
    price = base[c].replace(0, np.nan)

    with np.errstate(divide="ignore", invalid="ignore"):
        lr = np.log(price/price.shift(1))
    new_features[f"{c}_logret"] = lr.replace([np.inf, -np.inf], np.nan)

    new_features[f"{c}_vol"] = price.pct_change().rolling(14, min_periods=14).std()
    new_features[f"{c}_rsi"] = rsi_14(price)
```

สร้างฟีเจอร์พื้นฐานต่อสินทรัพย์

logret = log-return รายวัน $\log\left(P_t/P_{t-1}\right)$ กัน division-by-zero ด้วยแทน 0 เป็น NaN

vol = rolling 14D volatility (ส่วนเบี่ยงเบนมาตรฐานของ % change)

rsi = RSI-14 จากฟังก์ชันด้านบน

รวมเป็น feat_df = DataFrame(new_features, index=base.index)

```
# ----- BTC-specific -----
btc = base["BTC-USD"]
feat_df["BTC-USD_ma7"]      = btc.rolling(7,  min_periods=7).mean()
feat_df["BTC-USD_ma21"]     = btc.rolling(21, min_periods=21).mean()
feat_df["BTC-USD_momentum"] = btc / btc.shift(10) - 1
macd_fast = btc.ewm(12, adjust=False).mean()
macd_slow = btc.ewm(26, adjust=False).mean()
macd = macd_fast - macd_slow
feat_df["BTC-USD_macd"]   = macd
feat_df["BTC-USD_signal"] = macd.ewm(9, adjust=False).mean()
```

เส้นค่าเฉลี่ยเคลื่อนที่ 7 และ 21 วัน

โมเมนตัม 10 วัน = ผลตอบแทนสะสมช่วง 10 วัน

```
# ----- Cross-asset correlations (14D) -----
if all(x in base.columns for x in ["BTC-USD","^GSPC","GC=F","^IXIC"]):
    feat_df["SP500_corr"]  = base["BTC-USD"].rolling(14, min_periods=14).corr(base["^GSPC"])
    feat_df["GOLD_corr"]   = base["BTC-USD"].rolling(14, min_periods=14).corr(base["GC=F"])
    feat_df["NASDAQ_corr"] = base["BTC-USD"].rolling(14, min_periods=14).corr(base["^IXIC"])
```

วัดความสัมพันธ์เชิงเส้นระยะสั้นระหว่าง BTC กับสินทรัพย์หลักอื่นๆ

```
# ----- Regime features: 30D realized vol ของ BTC -----
btc_vol30 = btc.pct_change().rolling(30, min_periods=30).std()
feat_df["BTC_vol30"] = btc_vol30
q1, q2 = btc_vol30.quantile([0.33, 0.66])
feat_df["regime_low"]  = (btc_vol30 <= q1).astype(float)
feat_df["regime_mid"]  = ((btc_vol30 > q1) & (btc_vol30 <= q2)).astype(float)
feat_df["regime_high"] = (btc_vol30 > q2).astype(float)
```

ใช้ realized vol 30 วันแบ่งเป็น 3 โหมดความผันผวน โดยใช้ quantiles ที่ 33% และ 66%

สร้างเป็น one-hot regimes: low / mid / high (ค่า 0/1)

Feature engineering (Multi-asset + Regime)

เลือกฟีเจอร์ + train/val split + scaling ตรงนี้คือ เตรียมฟีเจอร์, แบ่ง train/val, normalize ป้องกัน data leakage เปลี่ยนเป็นลำดับเวลา (sequence) เอา time-series ทั้งก้อนมาทำเป็น sliding window ยาว 128 วัน target คือ label ของวันถัดจาก sequence

```
class SeqDataset(Dataset):
    def __init__(self, frame, fcols, seq_len=128):
        X, y = [], []
        vals = frame[fcols].values
        tars = frame["target"].values.astype(np.float32)
        for i in range(len(frame) - seq_len):
            X.append(vals[i:i+seq_len])
            y.append(tars[i+seq_len])
        self.X = torch.tensor(np.array(X), dtype=torch.float32)
        self.y = torch.tensor(np.array(y), dtype=torch.float32).unsqueeze(1)
    def __len__(self): return len(self.X)
    def __getitem__(self, idx): return self.X[idx], self.y[idx]
```

แต่ละตัวอย่างคือช่วงเวลา [i, i+seq_len-1] และ label คือค่า target ที่เวลา i+seq_len

สอดคล้องกับนิยาม target ก่อนหน้า (เช่น shift(-H))

GRU model (regularized)

```
class GRUModel(nn.Module):
    def __init__(self, in_dim, hidden=256, layers=3, dropout=0.5):
        super().__init__()
        self.gru = nn.GRU(
            input_size=in_dim,
            hidden_size=hidden,
            num_layers=layers,
            batch_first=True,
            dropout=dropout if layers > 1 else 0.0
        )
        self.drop = nn.Dropout(dropout)
        self.out  = nn.Linear(hidden, 1)  # binary logit
```

สถาปัตยกรรม GRUModel(in_dim, hidden=256, layers=3, dropout=0.5)

- ชั้นหลักคือ nn.GRU รับเวกเตอร์ฟีเจอร์ต่อเวลา in_dim
- ซ่อน hidden=256 จำนวนเลเยอร์ layers=3
- dropout ภายใน GRU จะทำงานเฉพาะกรณี layers>1 (ระหว่างเลเยอร์เท่านั้น ไม่ได้ดรอประหว่าง time steps)
- self.drop = nn.Dropout(0.5) หลังดึงสถานะท้ายลำดับ
- self.out = nn.Linear(256, 1) ส่งออก **logit** ตัวเดียวสำหรับ binary classification

```
  def forward(self, x):
      h, _ = self.gru(x)           # (B, T, H)
      h_last = h[:, -1, :]         # (B, H)
      h_last = self.drop(h_last)
      return self.out(h_last)      # (B, 1)
```

forward และมิติข้อมูล forward(self, x):

ใช้ representation ของ timestep สุดท้ายเป็นสรุปลำดับทั้งหมด แล้วแมปเป็น logit เดียว

การใช้งานกับ loss/metrics

เอา logit เข้า BCEWithLogitsLoss โดยตรง (ไม่ต้อง sigmoid ก่อน)

เวลาประเมินค่า probability ให้ใช้ torch.sigmoid(logit)

เหตุผลของดีไซน์นี้

GRU 3 เลเยอร์ช่วยเรียนรู้ลำดับลึกขึ้น (dependency ยาวขึ้น)

ใช้ h[:, -1, :] พอเพียงสำหรับทำนายสัญญาณที่อิงบริบทล่าสุด

Dropout หลัง timestep สุดท้ายช่วยลด overfit โดยไม่ทำให้ temporal dynamics ปน noise มากเกินไป

Train / Evaluate utils

```
def eval_model(model, loader, threshold=0.5):
    model.eval()
    y_true, y_prob, tot = [], [], 0.0
    loss_fn = nn.BCEWithLogitsLoss()
    with torch.no_grad():
        for xb, yb in loader:
            xb, yb = xb.to(device), yb.to(device)
            logit = model(xb)
            loss  = loss_fn(logit, yb)
            tot  += loss.item()
            prob  = torch.sigmoid(logit).cpu().numpy().ravel()
            y_prob.extend(prob)
            y_true.extend(yb.cpu().numpy().ravel())
    y_true = np.array(y_true); y_prob = np.array(y_prob)
    y_pred = (y_prob > threshold).astype(int)
    # กันเคส all-one/all-zero
    if len(np.unique(y_true)) < 2:
        f1, auc, acc = 0.0, 0.5, (y_pred==y_true).mean()
    else:
        f1  = f1_score(y_true, y_pred, zero_division=0)
        auc = roc_auc_score(y_true, y_prob)
        acc = (y_pred==y_true).mean()
    return tot/len(loader), f1, auc, acc, y_true, y_prob
```

eval_model(model, loader, threshold=0.5):

สลับโมเดลเป็นโหมดประเมินและปิดการคำนวณกราดิเอนต์ จากนั้นวนอ่านข้อมูลเป็นแบตช์ คำนวณลอจิตและ loss แบบ BCEWithLogitsLoss แปลงลอจิตเป็นความน่าจะเป็นด้วย sigmoid แล้วตัดสินคลาสด้วย threshold 0.5 รวบรวมค่าจริง (y_true) และความน่าจะเป็น (y_prob) เพื่อคำนวณ F1, AUC, ACC และหา

ค่า loss เฉลี่ย ถ้าป้ายกำกับมีเพียงคลาสเดียวในชุดประเมิน จะกำหนดค่าเริ่มต้นที่ไม่พัง (เช่น AUC=0.5) แล้วส่งคืนผลทั้งหมดพร้อม y_true, y_prob สำหรับใช้วิเคราะห์ต่อ

```
def train_model(model, train_loader, val_loader, epochs=200, lr=2e-4, patience=10):
    opt = torch.optim.AdamW(model.parameters(), lr=lr, weight_decay=1e-4)
    scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(opt, T_max=max(20, epochs//2), eta_min=1e-6)
    loss_fn = nn.BCEWithLogitsLoss()

    best_val = np.inf
    patience_ctr = 0
    history = {"train_loss": [], "val_loss": [], "f1": [], "auc": [], "acc": []}

    for ep in range(1, epochs+1):
        model.train()
        total = 0.0
        for xb, yb in train_loader:
            xb, yb = xb.to(device), yb.to(device)
            logit  = model(xb)
            loss   = loss_fn(logit, yb)
            opt.zero_grad(); loss.backward(); opt.step()
            total += loss.item()
        scheduler.step()

        val_loss, f1, auc, acc, _, _ = eval_model(model, val_loader, threshold=0.5)
        history["train_loss"].append(total/len(train_loader))
        history["val_loss"].append(val_loss)
        history["f1"].append(f1)
        history["auc"].append(auc)
        history["acc"].append(acc)

        print(f"Epoch {ep:03d}: train={total/len(train_loader):.4f}  val={val_loss:.4f}  F1={f1:.3f}  AUC={auc:.3f}  ACC={acc:.3f}")

        # early stopping ตัดสินด้วย val_loss
        if val_loss < best_val - 1e-4:
            best_val = val_loss
            patience_ctr = 0
            best_state = {k: v.detach().cpu().clone() for k, v in model.state_dict().items()}
        else:
            patience_ctr += 1
            if patience_ctr >= patience:
                print(f"Early stopping at epoch {ep}")
                break

    if 'best_state' in locals():
        model.load_state_dict(best_state)
    return model, history
```

train_model(model, train_loader, val_loader, epochs=200, lr=2e-4, patience=10):

สร้างออปติไมเซอร์ AdamW และตัวลดอัตราเรียนรู้แบบ Cosine Annealing แล้ววนเทรนตามจำนวน epoch ในโหมดฝึก: คำนวณลอจิต, loss, ทำ backprob และอัปเดตพารามิเตอร์ เสร็จหนึ่ง epoch จึงประเมินบนชุดตรวจสอบด้วย eval_model เก็บสถิติ train_loss, val_loss, F1, AUC, ACC และพิมพ์ผล Early Stopping จะเฝ้าดู val_loss ถ้าไม่ดีขึ้นตามเกณฑ์ในช่วง patience จะหยุดก่อนและโหลดพารามิเตอร์ที่ดีที่สุดกลับมา สุดท้ายคืนโมเดลที่ดีที่สุดพร้อมประวัติการเทรน (history)

ส่งลิงค์ที่นำไปสู่โค้ด PyTorch หรือ TensorFlow ที่ทุกคนเข้าถึงได้มาด้วย ควรจะเป็น Github public repo link

https://github.com/tirenton/DL_Final

6. อธิบายวิธีในการ train ตัว deep learning network ที่เลือกมาใช้ รวมไปถึงอธิบาย dataset ที่เกี่ยวข้องและแหล่งที่มา

หัวข้อนี้ผมเทรนและประเมินโมเดลแบบเป็นขั้นตอนชัดเจนตั้งแต่เตรียมข้อมูลจนถึงเลือกเกณฑ์ตัดสินผลลัพธ์ ข้อมูลดึงจาก API yfinance ของ Yahoo Finance โดยใช้ราคาปิดที่ปรับแล้วของหลายสินทรัพย์พร้อมกัน ได้แก่ BTC-USD, ^GSPC, GC=F และ ^IXIC ช่วงเวลาเริ่มจาก 2018-01-01 ถึงวันที่ดึงจริง เป้าหมายเป็นงานจำแนกแบบทวิภาค โดยให้ y = 1 ถ้าราคา BTC ในอีก 30 วันข้างหน้าสูงกว่าวันนี้ และ y = 0 ถ้าไม่สูงกว่า เพื่อให้เห็นมุมมองข้ามตลาด ผมทำฟีเจอร์ทั้งเชิงเทคนิคและความสัมพันธ์ระหว่างสินทรัพย์ไว้ครบก่อนเข้าโมเดล

ฝั่งฟีเจอร์ ผมสร้าง log-return, volatility แบบ rolling, RSI, เส้นค่าเฉลี่ย MA7/MA21, momentum, MACD และ signal line ของ MACD แล้วเสริม cross-asset correlation ระหว่าง BTC กับ S&P500, Gold และ NASDAQ เพื่อให้โมเดลจับลิงก์ระหว่างตลาดได้ นอกจากนี้มี regime ของความผันผวนแปลงเป็น one-hot ว่าอยู่โหมดนิ่ง กลาง หรือผันผวนสูง ทั้งหมดนี้สเกลด้วย StandardScaler โดย fit เฉพาะบน training set แล้วค่อย transform validation เพื่อลด data leakage จากนั้นผมจัดข้อมูลให้อยู่ในรูป time window ต่อหนึ่งตัวอย่าง เพื่อป้อนเข้าโมเดลลำดับเวลา

ตัวโมเดลเป็น GRU 3 ชั้น ขนาดซ่อน 256 หน่วย พร้อม dropout 0.5 เอา hidden state ของวันสุดท้ายไปเข้าชั้นเชิงเส้นเพื่อได้ logit หนึ่งค่า ใช้ฟังก์ชันสูญเสียแบบ BCEWithLogitsLoss ซึ่งสะดวกตรงที่รวม sigmoid เข้าไว้ในตัว ทำให้เสถียรเรื่องตัวเลขตอนเทรน ออปติไมเซอร์ใช้ AdamW และมี CosineAnnealingLR ช่วยปรับ learning rate ให้นุ่มลงช่วงท้าย ๆ ผมเทรนด้วย batch size 64 สูงสุด 200 epochs แต่มี early stopping ดูจาก val_loss ถ้าไม่ดีขึ้นเกิน 10 epochs จะหยุดและโหลด weight ที่ดีที่สุดกลับมา

การแบ่งชุดข้อมูลทำแบบตามเวลา 80/20 ไม่สลับลำดับ เพื่อไม่ให้อนาคตไปรั่วใส่อดีต จากนั้นผมวัดผลบน validation ทุก epoch ทั้ง F1, AUC และ Accuracy โดยพยากรณ์เป็นความน่าจะเป็นแล้วค่อยตัดสินใจด้วย threshold เริ่มที่ 0.5 นอกจากนี้ยังสแกนหลาย threshold เพื่อหา best-F1 ที่เหมาะกับ class balance ของชุดข้อมูล พร้อมวาด ROC และ Precision-Recall เพื่อดูภาพรวมการแลกเปลี่ยนระหว่าง precision กับ recall ให้ชัด

โดยรวม กระบวนการเทรนของผมตั้งใจคุมสามเรื่องหลัก หนึ่ง คุณภาพฟีเจอร์ที่ครอบคลุมทั้งสัญญาณเดี่ยวและสหสัมพันธ์ข้ามสินทรัพย์ สอง วินัยทางเวลาในการ split และ scaling เพื่อกัน leakage สาม เสถียรภาพระหว่างเทรนด้วย AdamW, CosineAnnealingLR และ early stopping ส่วนการรายงานผลก็ให้ครบทั้งตัวเลขและกราฟ ทำให้เห็นทั้งภาพรวมเชิงสถิติและพฤติกรรมของโมเดลบนเส้นโค้ง ROC/PR ว่าพอ

เปลี่ยนเกณฑ์ตัดสินแล้วสมดุล precision-recall ขยับไปทางไหนบ้าง ซึ่งช่วยสะท้อนว่าโมเดลพร้อมใช้งานในบริบทจริงมากแค่ไหนครับ

7. อธิบายการประเมิน (evaluate) model แสดงค่า loss จากการ train และ metric ที่เหมาะสมในการประเมิน เช่น accuracy precision หรือ recall

เกณฑ์วัดผล (Metrics)

ประเมินผลด้วยสามตัวชี้วัดหลักบนชุดประเมิน

- Loss ใช้ BCEWithLogitsLoss ตรวจสอบการลู่เข้าระหว่างเทรนและวาลิเดชัน ถ้าเส้น val loss เริ่มทรงตัวหรือสูงขึ้นขณะ train loss ลดลง แปลว่ามีโอกาสเกิด overfitting
- AUC วัดความสามารถในการจัดอันดับความน่าจะเป็นทิศทางขึ้นลงโดยไม่ยึดติด threshold เหมาะกับชุดข้อมูลที่ class balance อาจไม่เท่ากัน
- F1 และ Accuracy รายงานคู่กัน เพื่อเห็นสมดุล Precision-Recall กับสัดส่วนทายถูกภาพรวม โดย F1 ใช้เมื่อสนใจความคุ้มค่าของสัญญาณที่ยืนยันแล้วมากกว่าแค่เปอร์เซ็นต์ทายถูก

การเลือกค่าเกณฑ์ตัดสิน (Threshold Selection)

โมเดลพยากรณ์เป็นความน่าจะเป็นจาก sigmoid บน logit แล้วแปลงเป็นคลาสด้วย threshold ค่าเริ่มต้นที่ 0.5 และทำการกวาดค่าหลายระดับเพื่อหา threshold ที่ให้ F1 สูงสุดบนชุดประเมิน จากนั้นรายงานตัวเลขที่ threshold ดีสุดร่วมกับ AUC ที่ไม่ขึ้นกับ threshold เพื่อความโปร่งใส

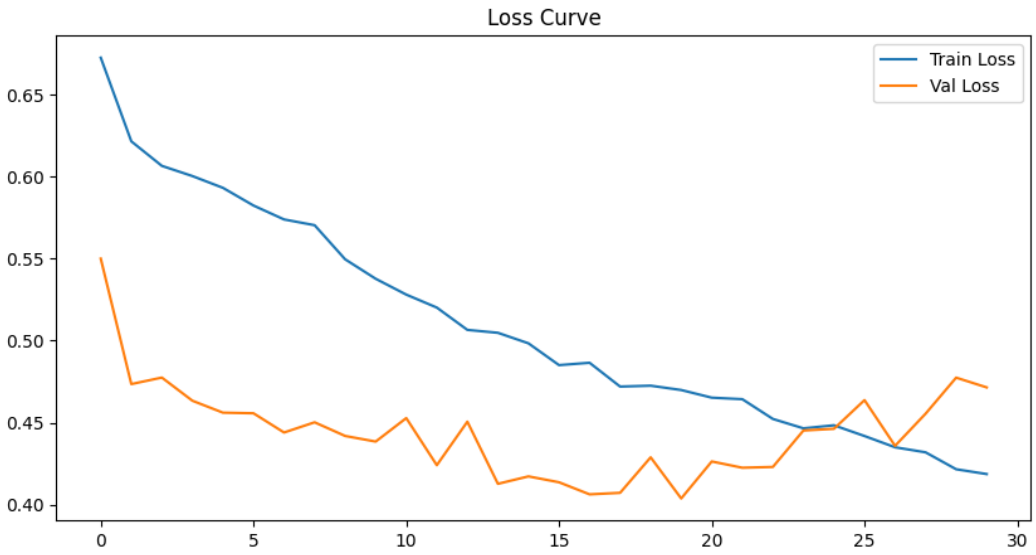
การหยุดก่อนเวลาและการเลือกโมเดล (Model Selection)

ใช้ early stopping โดยยึด val loss เป็นตัวตัดสิน ถ้าไม่ดีขึ้นต่อเนื่องเกิน 10 epochs จะหยุดเทรนและโหลดเวทที่ดีที่สุดกลับมา ช่วยป้องกัน overfitting และลดเวลาเทรนโดยไม่เสียคุณภาพ

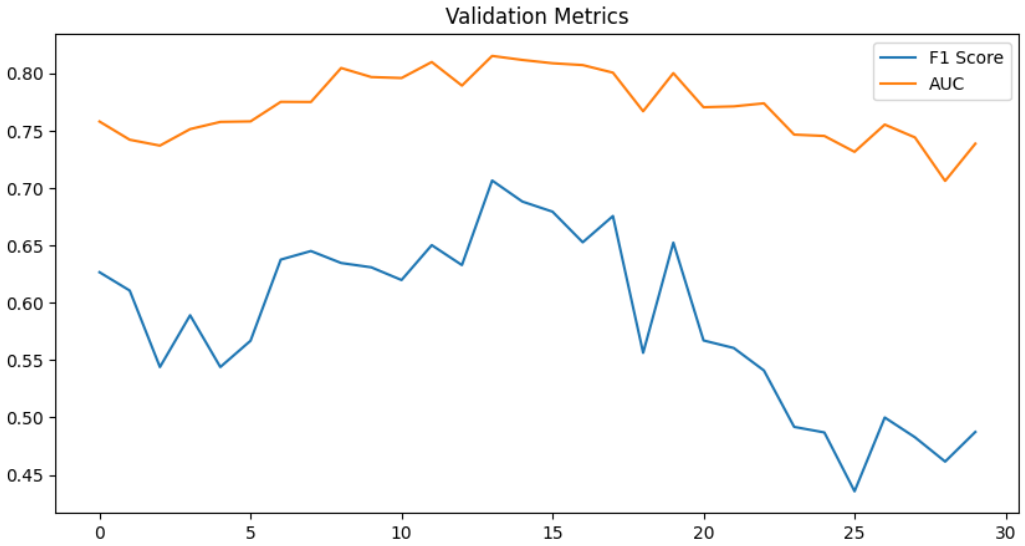
กราฟและตารางประกอบ (Visualization)

รายงานกราฟและตารางดังนี้

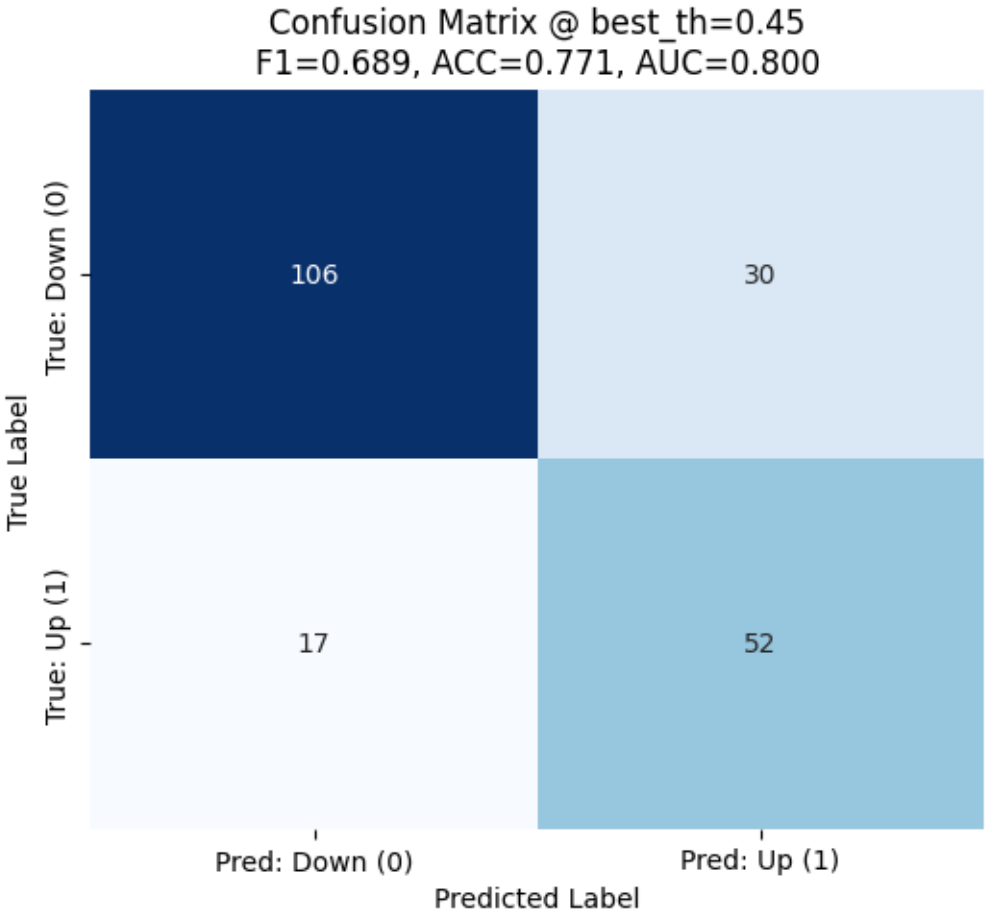
- Loss Curves เส้น train loss และ val loss ต่อ epoch เพื่อดูการลู่เข้าและจุดเริ่ม overfit



- Validation Matrix (F1 Score and AUC)



- Confusion Matrix ที่ threshold ดีสุด เห็นจำนวน TP FP TN FN ชัดเจน



ผลลัพธ์ของตัวที่ดีที่สุด

ผลลัพธ์โมเดลตัวที่ดีดีที่สุดที่ทำการบันทึกไว้แล้วทำการ Re-Evaluate หลังจากได้ค่า Threshold เพื่อที่จะทำให้ค่า F1 Optimized

TN (True Negative)	106	ทำนายถูกฝั่งลบเยอะ
FP (False Positive)	30	ผิดฝั่งบวกเล็กน้อย
FN (False Negative)	17	พลาดบวกน้อย
TP (True Positive)	52	จับ positive ได้ดี
Precision	0.634	ความแม่นเมื่อทายว่าขึ้น
Recall	0.754	ความสามารถในการจับเหตุการณ์ขึ้นได้ครบ
F1	0.689	สมดุลระหว่าง Precision และ Recall
Accuracy	0.771	ทายถูกทั้งหมด 77.1%
AUC	0.800	จำแนกแนวโน้มขึ้น/ลงได้ชัดเจนดีมาก

ผลนี้ชัดเจนว่า โมเดลสามารถจำแนกทิศทางราคาที่จะขึ้น หรือไม่ขึ้น ได้อย่างมีประสิทธิภาพ โดยให้ค่า AUC = 0.800 ซึ่งถือว่าดีมาก แสดงว่ามีความสามารถในการแยกคลาสได้อย่างชัดเจน นอกจากนี้ยังมี Accuracy = 0.771 และ F1 = 0.689 สะท้อนว่าโมเดลมีสมดุลที่ดีระหว่างการตรวจจับเหตุการณ์บวก (Recall = 0.75) และความแม่นยำในการทำนายบวก (Precision = 0.63)

แม้ข้อมูลจะมีความไม่สมดุล (Positive ratio ≈ 0.33) แต่โมเดลยังคงให้ผลลัพธ์ที่เสถียร และเหมาะสมสำหรับการนำไปใช้ทำนายแนวโน้มราคาในอนาคต โดยเฉพาะระยะกลาง (เช่น 30 วัน) ซึ่งให้ผลดีกว่าการทำนายระยะสั้นที่มีความผันผวนสูง

8. อธิบายบทความอ้างอิงและงานที่เกี่ยวข้อง

แหล่งข้อมูลและฟีเจอร์ทางการเงิน

- yfinance API: ใช้ดึง Adjusted Close จาก Yahoo Finance ได้สะดวกและเป็นมาตรฐานในงานวิจัยเชิงปฏิบัติ ช่วยให้ reproducible กับ universe หลายสินทรัพย์
- RSI: อินดิเคเตอร์โมเมนตัมที่วัดความแรงของขาขึ้น-ลง สอดคล้องกับการสร้าง rsi(14) ในโค้ด
- Murphy, Technical Analysis: อธิบายการใช้ MA, MACD, momentum ซึ่งใช้เป็นฟีเจอร์หลัก
- Moskowitz et al., Time-Series Momentum: สนับสนุนการใช้ log-return และโมเมนตัมเป็นสัญญาณพื้นฐาน

สถาปัตยกรรมลำดับเวลา (RNN/GRU) สำหรับการคาดการณ์การเงิน

- Hochreiter & Schmidhuber, LSTM: วางพื้นฐาน RNN ที่จำระยะยาว ชี้ปัญหา vanishing gradient ที่สถาปัตยกรรม gated แก้ได้
- Cho et al., GRU: เสนอ GRU ที่เบากว่า LSTM แต่เก็บบริบทระยะยาวได้ดี สอดคล้องกับการใช้ GRU 3 ชั้นในโค้ดของผม
- Fischer & Krauss, LSTM for Stock Returns: RNN กับข้อมูลการเงินเชิงลำดับ ช่วยอ้างอิงว่าการใช้โมเดล sequence เหมาะกับโจทย์นี้

Regime, ความผันผวน และพลวัตความสัมพันธ์

- Hamilton, Regime-Switching Models: แนวคิดตลาดหลายระบอบ ช่วยอธิบายการทำ one-hot regime จาก 30-day realized vol
- Andersen et al., Realized Volatility: รองรับการใช้ความผันผวนแบบ rolling จากผลตอบแทนรายวัน

- Ang & Bekaert, International Asset Correlations: ยืนยันว่าความสัมพันธ์เปลี่ยนตามสภาวะตลาด สนับสนุนฟีเจอร์ rolling correlation
- Longin & Solnik, Extreme Correlations: พบว่าความสัมพันธ์เพิ่มขึ้นในช่วงวิกฤต สอดคล้องกับการใส่ฟีเจอร์ทั้ง correlation และ volatility regime

การเทรนและการประเมินผลสำหรับ Binary Directional Forecasting

- Paszke et al., PyTorch: กรอบการทำงานของโค้ดโมเดล GRU
- Loshchilov & Hutter, AdamW: ออปติไมเซอร์ที่แยก weight decay ออกจากการประมาณโมเมนตัม
- BCEWithLogitsLoss: แนวปฏิบัติมาตรฐานสำหรับ binary classification ที่ประกอบ sigmoid ภายในเพื่อเสถียรภาพทางตัวเลข
- Loshchilov & Hutter, Cosine Annealing: เทคนิคลด learning rate แบบโค้งคอไซน์ที่ช่วย fine-tune ช่วงท้าย ใช้ผ่าน CosineAnnealingLR
- Saito & Rehmsmeier, Precision-Recall: แนะนำการดู PR และการเลือก threshold ให้เหมาะกับ class balance สอดคล้องกับขั้นตอนสแกน threshold หายอด F1
- Fawcett, ROC Analysis: พื้นฐานการอ่าน ROC-AUC
- De Prado, Advances in Financial Machine Learning: แนวคิดการประเมินผลที่คำนึงถึงความลื่นไหลของตลาดและความระมัดระวังด้าน backtest สอดคล้องกับภาพรวมวิธีคิดของโปรเจกต์

9. ถ้าทำเป็นกลุ่ม บอกงานที่แต่ละคนทำและให้สัดส่วนเป็นเปอร์เซ็นต์ของงานทั้งหมด

งานที่ทำ	สัดส่วน	อภิภู	อิทธิเดช
ริเริ่มไอเดียและออกแบบโจทย์	10%		✓
จัดหา-เตรียมข้อมูลและ Data Pipeline	15%		✓
Feature Engineering และ Regime Features	20%	✓	✓
พัฒนาโมเดลหลัก (GRU) และโค้ดเทรน	20%	✓	✓
การประเมินผล	15%	✓	
Visualization และสรุปผลการทดลอง	10%	✓	
เอกสารรายงาน	10%	✓	✓
เปอร์เซ็นต์ทั้งหมด	100%	50%	50%